Web Services Composition: Mashups Driven Orchestration Definition

Sébastien Mosser, Franck Chauvel, Mireille Blay-Fornarino, Michel Riveill
University of Nice – Sophia Antipolis,
CNRS, I3S Laboratory, RAINBOW team,
Sophia Antipolis, France
{mosser,blay,riveill}@polytech.unice.fr
franck.chauvel@i3s.unice.fr

Abstract

*On the one hand, mashups are a new kind of web application built upon the composition of different resources in a user-friendly way. Tools based on such concepts focus on graphic design and allows final users to build complex applications using pipes to connect data sources into a data–flow. It underlines a constant need for making services resuable in an easy way. On the other hand, Web Services Oriented Architecture (*WSOA*) supports development of high quality applications based on a control–flow between services. We explore in this paper how a* WSOA *can be defined as a data-flow in a mashup-like approach, where Model Driven Engineering techniques enable a clever composition of data-flows and the generation of control-flows based architecture.*

1. Introduction

One of the major trend in web-based system design is to use graphical programming environment. Mashups [24] for instance allow web-system designers (*"web architect"*) to graphically combine web content and services, producing new features for their system [5]. For Human-Computer Interaction community, *"mashups allow end-user programming for the Web"* [26]. Mashups design fits biologist needs [3] as well as semantic web experiments [6]. From a technical point of view, mashups rely on dedicated platforms, like JOPERA [21] as a dedicated mashup application server or shared mashup platform[1].

On the opposite Web Service Oriented Architectures (WSOA) provides a way to implement scalable Services Oriented Architectures (SOA, [13]) using web services as elementary services, and orchestrations [22] as composition mechanisms. The W3C defines orchestrations as "*the pattern of interactions that a Web Service agent must follow in order to achieve its goal*" [25]. Specialized (*i.e. elementary*) code is written inside web services, and each business process is described as an orchestration of those web services.

The contribution of this paper is to fill the gap between mashups and orchestrations, as the mashup emergence shows that the WSOA methodology is not yet the abstraction level that a web architect needs. Then our contribution proposes to take advantages of the model-driven approach [11] by capturing mashups and WSOA dialects into dedicated meta-models. Thus, we propose to make an automatic transformation between models which conform to these two meta-models.

The remainder of the paper is organized as follow. The next section uses an academic information system case study (named SEDUITE) to illustrate why we need to automatically produce WSOA from mashups. Section 3 describes the two meta-models used to capture both web architects' and WSOA designers' dialect. We propose in Section 4 a transformation and apply it on our example. Tools support is exposed in Section 5. After having presented a selection of relevant contributions on this area in Section 6, we conclude the paper showing perspectives of mashup to WSOA transformations.

2. Mashups examples: SEDUITE

SEDUITE is an information system especially designed for academic institutions. It aims to retrieve and then broadcast "scholar" information (events, timetable) to students and teachers. Based on a WSOA, it exposes information sources as *services* and uses *orchestrations* to retrieve and then compose information. Information is accessed via different physical devices like public *plasma screen* or private *personal digital assistant* (PDA). More information about

[1] http://pipes.yahoo.com, http://www.popfly.ms, ...

SEDUITE can be found on the project website[2].

2.1. Mashup syntax & Semantic

We use a graphical syntax to represent mashups. As there is no real consensus on graphical mashups formalism, we define a syntax inspired by McGraw VAL data-flow language, defined in [14]. *Rectangles* represent sources of informations, *circles* represent filters applied on informations and *direct arcs* represent information path between those nodes. *Triangles* represent nodes' inputs with associated name, and quoted text means constant input. The resulting graph represents a *mashup*. This simple graphical syntax fosters final user friendliness, massively used in mash-up tools.

The execution of a graph is based solely on operand availability: each node may begin execution as soon as all input are present. When a node completes, results are transmitted via the output arcs to next node(s).

2.2. Information mashups

FIG 1 shows how public school information can be composed before being displayed on public screen. `Weather`, `Timetable` and `News` are sources of information implemented as legacy services. We retrieve from the weather forecast service information about school city (here *"Nice, FR"*), and filter timetable events on classes physically present in our building (here CS_3, CS_4, CS_5). Global news, filtered events and weather forecast are then concatenated (using '+') and returned. This information flow is considered as a new available information source called `SchoolInfo`.

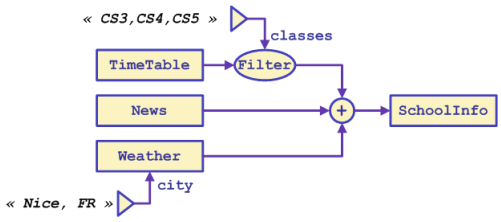

Figure 1. $SchoolInfo$ **mashup**

School users (students, teachers, administrative staff, ...) are interested in more personalized information [9]. The mashup in FIG 2 defines how personal information can be retrieved from the system. A user expresses a *profile* (defined as a set of values representing services parameters), connect this profile to information sources. User's personal informations (friends timetable events and weather forecast information of her living place) are then broadcasted with usual informations when she use this mashup.

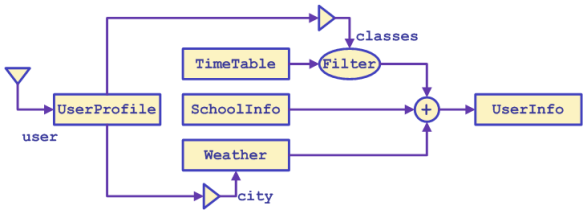

Figure 2. $UserInfo$ **mashup**

2.3. Goal: reaching an existing system

It seems natural for SEDUITE designers to express information flows as arrows between boxes. But in order to fit with underlying legacy and scalable infrastructure, a designer can only deal with orchestrations of web services. So, she must *(i)* transform by hand a data-flow (design domain) into a control-flow implementation to find the entry point of the expected orchestration, *(ii)* express this orchestration using a specialized language and *(iii)* take into account existing services constraints.

Considering the previously defined mashups as two different versions of the same functionality (*ie* retrieve information), we aim to generate a single orchestration called `InformationProvider` which provides these two business operations. This orchestration will have to call existing services (`News`, `Weather`, ...). Recurrent *sub-processes* (like filtering timetable events) can be identified as a business functionality and then provided as an operation of another orchestration focused on timetable management.

3. Sketching meta-models

In order to support such an automatic transformation process, we define two meta-models to capture domain-specific expression capability. This section presents the transformation source and target meta-models.

3.1. Source: Mashups data-flow

Our meta-model (FIG 3) defines a `Flow` as a set of `Nodes` and `DataPaths` to connect `Nodes`.

Inspired by Garlan [2] who defines *Pumps* and *Filters*, this meta-model specializes the `Node` concept into *(i)* `Sources` and *(ii)* `Processes`. A `Source` only provides output `Slots`, where a `Process` applies a `function` on input `Slots` and provides output `Slots`.

[2] `http://anubis.polytech.unice.fr/jSeduite`

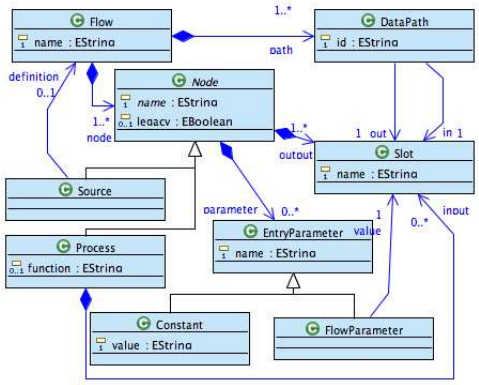

Figure 3. Mashup data-flow meta-model

A `Node` can accept some parameters: *(i)* `Constants`, *(ii)* `FlowParameters` where parameter values comes from an output `Slot` and *(iii)* `EntryParameters` valued at runtime by final user.

3.2. Target: WSOA control-flow

We voluntary define a reduced WSOA meta-model in FIG 4, as this contribution focus on the architectural transformation of a data-flow (mashup) into a control-flow (orchestration).

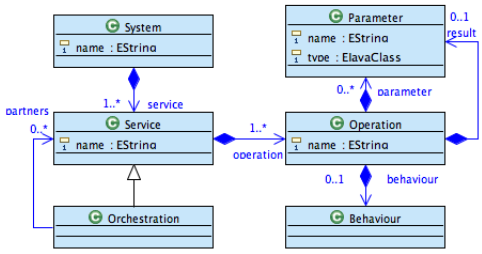

Figure 4. Reduced WSOA meta-model

A `System` is defined as a set of `Services`. Each `Service` defines `Operations`, accepting several `Parameters` as input. An `Operation` can return a `Parameter` as output value. `Orchestrations` specialize the `Service` concept, referencing other `Services` as `partners`. An `Operation` is considered by default as a black box, but can be enriched by a `Behaviour`, expressed in an external formalism such as WSBPEL (Web Services Business Process Execution Language, [10]) for example.

4. Transformation: Reaching the WSOA

The goal of this transformation is to generate an adequate WSOA from a set of mashups. Different mashups refering to a same functionnality are grouped in an orchestration where each mashup corresponds to an operation. For each operation, we have to generate a control-flow, *ie* identify perameters of operations, optimize control flow avoiding useless service calls, and identify some parts of the control flow as new services. We propose here a methodology which first reason on the data-flow before transform it into a control–flow.

4.1. Data–flow reasoning & Optimization

We identify several graph rewriting techniques T_x which allow us to optimize the data–flow as the first step of the transformation.

- $\mathbf{T_0}$: *"Unfolding invocation"*. If reusing mashups is natural at design level, it leads to redundant service invocations when implementing it as a control-flow. So the first step consists in *"unfolding"* mashups. Unfolding operation (*ie* replace each mashup by its definition inside others mashups) allows a global reasoning on the whole data-flow structure. Architecture in FIG 2 is easier to understand, design and adapt than the one in FIG 5. However the last one is a better base model for optimization and generation of a WSOA.

- $\mathbf{T_1}$: *"Grouping invocation"*. When working on unfolded data-flows, overlapping invocation and invocation sequences can appear. Some optimization can then semi-automatically be performed to *"group"* these invocations into a single one or to retract some. In case of need, it can involve the data-flow designer [19]. For instance, as WSOA fundamentally relies on stateless Web Services, different references to a same source of information with the same parameters gives the same result and can be avoided. In a general case, we established a compositional algebra based on the semantics of the data-sources and filters.

- $\mathbf{T_2}$: *"Identifying business operations"*. When a data-path is recognized several times from different mashups after T_1 usage, we identify it as a business operation which should be exposed on its own and shared among others control-flow. The identification is based on maximum-length sequence recognition. Identifying recurrent data paths [23] through multiple expressed mashups eases the architect work of identifying control-flow granularity. Software refactoring community [1] uses similar techniques to extract recurrent methods from existing source code.

4.2. Control-flow generation

Based on the resulting set of data-flows, we now generate control-flows conforming to the WSOA meta-model. First of all, we identify in the mashup data-flows legacy services as fixed points during the transformations. Theses nodes are binded to black-box `Services` and will be used as `Partners` in the targeted architecture. Then, we perform an inversion[3] of the data-flow to identify entry points of the control flow as graph source[4]. Each identified entry point is mapped to an operation of the orchestration and then retracted (*ie* retracting the `Node` and all `DataPath` using it) from the inverted mashup. We iterate over the mashup node set and stop the loop when all entry points are fixed points.

`EntryParameters` of the original mashup are handled through the data-path and rises as `Parameters` of generated `Operations`. Others flow parameters are reified inside the behaviour of the resulting `Orchestration`: *(i)* `Constants` are generated as WSOA behaviour constants and *(ii)* `FlowParameters` as partners invocation.

Due to previous section reasoning, the resulting control-flow is not only a valid WSOA in term of meta-model structural conformity, but it conforms too to usual guidelines of SOA [20].

4.3. Example: Transforming SEDUITE

We consider here the transformation of both *UserInfo* and *SchoolInfo* mashups to generate `InformationProvider` orchestration. We focus here on *UserInfo*, as *SchoolInfo* is still optimum from our point of view.

First of all, we unfold (T_0) $SchoolInfo$ inside the mashup, and obtain FIG 5 mashup. We apply the sequence

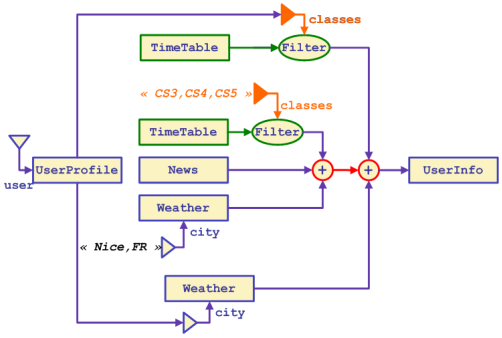

Figure 5. $Unfold(UserInfo)$

recognition to trigger the invocation grouping technique T_1,

[3] $(A \rightarrow B)^{-1} \equiv (A \leftarrow B)$

[4] $s \in Sources \Rightarrow indegree(s) = 0$

and identify three matching points: *(i)* "get timetable and then filter data" sequences, *(ii)* "get weather" redundant invocations and *(iii)* redundant usage of the `'+'` concatenation function. The following list details the optimization step:

1. Redundant `TimeTable` invocations are replaced by a single one. As `Filter` is a known function extracting information conforming to parameters we automatically merge them using the union (represented as $\cup$ symbol) of previous parameters.

2. Weather service is a legacy one. It only accepts a single parameter, so there is no composition rule to apply.

3. The concatenation sequence is replaced by a single concatenation of previous parameters where redundant parameters are retracted.

The resulting mashup $UserInfo'$ is shown in FIG 6. We can now analyze the two mashups together using T_2.

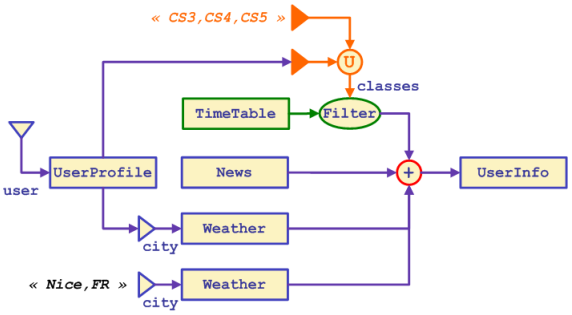

Figure 6. $UserInfo' \equiv Optimize(UserInfo)$

The path recognition technique identifies the timetable sequence as a business subprocess shared by $SchoolInfo$ and $UserInfo'$. After asking final user, it is transformed into a different orchestration shared by others.

We can now transform the mashup set into orchestration. The first identified graph sources are $UserInfo$ and $SchoolInfo$ nodes. As they belong to the same business silo (from user knowledge), they are generated as two operations `getSchoolInfo` and `getUserInfo` of an orchestration named `InformationProvider`. Another orchestration called `FilteredTimeTable` is generated following previous paragraph directive. Orchestrations partners and operation parameters are deduced from incident data-path in original mashups. FIG 7 shows the output of the transformation process using UML class diagrams ans stereotypes formalism.

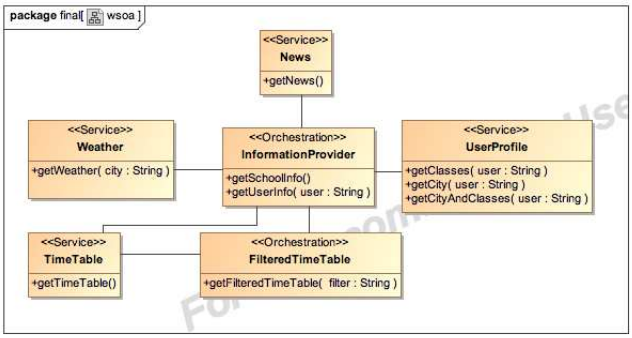

Figure 7. Transformation output

5. Implementation & Validation

We developed a first prototype of our approach using the ECORE [15] framework and the KERMETA langage [18]. The ECORE framework enables the creation of the two meta-models and of their related models. Models are stored as XMI files and can be manipulated by a wide range of model-transformation tools and languages. We use the reflexive ECORE's editor to build the data-flow models. Transformations are designed as a two steps process: first, a *Builder* object runs over the input model and builds the related output objects. Then, a *Linker* object goes through the input models and links together output objects. Each step of the method proposed here (reasoning and then WSOA generation) is implemented as a transformation following this method.

The FAROS[5] French national research project deals with SOA reliability. From a business domain model it allows the expression of contracts and reach several execution platforms as target. Considering the information diffusion as a business domain and SEDUITE as a validating application, the transformation proposed here is a subset of FAROS work and is naturally validated as a part of the project.

6. Related Work

Several visual notations have been proposed to specify workflow models with different expressive power, syntax and semantics. Usual data-flow languages express complex structures. These languages like VAL, LUSTRE [8] or SIGNAL [4] claims to represent a data-flow from A to Z. Our approach only uses data-flow expressiveness to design architecture and then reach legacy systems. Moreover, as it is a young experiments, we only address a reduced set of data-flow capabilities. We do not manage loops, XOR splits, error handling, ... Contrarily to [12], we do not extends a service model to represent mashups but express a fully dedicated business meta-model, following model-driven approach.

We propose an automatic transformation to reach legacy systems. In our implementation, mapping between high-level model and legacy system entities is based on name matching. This approach is quite naive but satisfying for experimental purpose. To reach legacy systems, a binding model can be defined to fill the gap between models and existent entities. The transformation process will then embed this model (expressed as a configuration file for example) and perform an efficient binding.

Function properties inference to automatically perform optimization is based on *"good properties"* of compositional algebra [19], but such properties are not reified in the given source meta-model. To ensure automatic composition, a property model must be defined, and a composition algebra could manage the effective composition.

Grid Computing community also expresses complex application as data-flow. They use domain-specific languages and execution engine (like SCUFL code interpreted using MOTEUR engine [16]) to express data-intensive application. Contrarily to our approach, dedicated grid infrastructure is a need for those algorithms which are often defined as massively parallel algorithm. Optimization techniques of grid workflows is a very productive research field [7].

Several studies address software evolutions, especially WSOA evolutions [17]. We consider the proposition described here as a good complement of these approaches. As expressing a data-flow is the natural dialect for designer, our model-driven approach captures a business domain. So, it could be possible to define evolutions at business level and then reach evolutions platforms using similar techniques.

7. Conclusions & Perspectives

Mashups systems are now a common tool to graphically design a web application as a simple data-flow process. Mashups hide most of the low-level complexity of web service development: the *Service Oriented* paradigm does not provide the relevant abstraction level. However the generation of a WSOA from a data-flow system is needed and currently still a hand-craft and error prone task.

The contribution of this paper is to perform a transformation between data-flow and service-oriented legacy architectures using model-driven engineering techniques. We provide two meta-models to capture both mashups systems and WSOA systems and a set of model transformations to enable the translation between those two meta-models. We provide a prototype of this transformation developed using the ECORE framework and the KERMETA language. We also apply this transformation to handle a WSOA designed for a school information broadcast system named SEDUITE.

Immediate perspectives of this work are to improve the

[5] http://www.lifl.fr/faros

reasoning model. As reasoning mechanisms presented here relies on logical programming techniques and definitions, the object oriented approach used by KERMETA is not optimal to express this kind of endomorphism. A usual workaround is to use dedicated tools (like PROLOG language) to implement this part of the transformation. Implementing the mapping between these tools must be done in a dedicated way and is still an ongoing work.

As a possible future work, we plan to apply this approach to larger data-flow systems such as systems deployed on cluster and grids. It would enable the automated generation of a set of jobs, which could be directly used on a grid.

References

[1] *Refactoring: improving the design of existing code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[2] R. B. Allen and D. Garlan. A formal approach to software architectures. In *IFIP Congress, Vol. 1*, pages 134–141, 1992.

[3] F. Belleau, M.-A. A. Nolin, N. Tourigny, P. Rigault, and J. Morissette. Bio2rdf: Towards a mashup to build bioinformatics knowledge systems. *Journal of biomedical informatics*, March 2008.

[4] A. Benveniste, P. L. Guernic, and C. Jacquemot. Synchronous programming with events and relations: the signal language and its semantics. *Sci. Comput. Program.*, 16(2):103–149, 1991.

[5] B. Bioernstad and C. Pautasso. Let it flow: Building mashups with data processing pipelines. In *Mashups'07 International Workshop on Web APIs and Services Mashups at ICSOC'07*, Vienna, Austria, 23/09/2007 2007.

[6] C. Bizer, R. Cyganiak, and T. Gauss. The rdf book mashup: From web apis to a web of data. In S. Auer, C. Bizer, T. Heath, and G. A. Grimnes, editors, *SFSW*, volume 248 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[7] T. Glatard, J. Montagnat, D. Emsellem, and D. Lingrand. A Service-Oriented Architecture enabling dynamic services grouping for optimizing distributed workflows execution. *Future Generation Computer Systems*, 24(7):720–730, July 2008.

[8] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data-flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[9] C. Joffroy, A.-M. Pinna-Déry, P. Renevier, and M. Riveill. Architecture model for personalizing interactive service-oriented application. In *11th IASTED International Conference on Software Engineering and Applications (SEA'07)*, pages 379–384, Cambridge, Massachusetts, USA, Nov. 2007. IASTED, ACTA Press.

[10] D. Jordan, J. Evedmon, A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guízar, N. Kartha, K. Liu, R. Khalaf, D. Konig, M. Marin, V. Mehta, S. Thatte, D. Van der Rijn, P. Yendluri, and A. Yiu. Web services business process execution language version 2.0. Technical report, OASIS, 2007.

[11] S. Kent. Model Driven Engineering. *Integratted Formal Methods. Third International Conference, IFM*, pages 15–18, 2002.

[12] X. Liu, Y. Hui, W. Sun, and H. Liang. Towards service composition based on mashup. In *Services, 2007 IEEE Congress on*, pages 332–339, 2007.

[13] M. MacKenzie, K. Laskey, F. McCabe, P. Brown, and R. Metz. Reference Model for Service Oriented Architecture 1.0. Technical Report wd-soa-rm-cd1, OASIS, Feb. 2006.

[14] J. R. McGraw. The val language: Description and analysis. *ACM Trans. Program. Lang. Syst.*, 4(1):44–82, 1982.

[15] E. Merks, R. Eliersick, T. Grose, F. Budinsky, and D. Steinberg. *The Eclipse Modeling Framework*. Addison Wesley, 2003.

[16] J. Montagnat, D. Jouvenot, C. Pera, Á. Frohner, P. Kunszt, B. Koblitz, N. Santos, and C. Loomis. Bridging clinical information systems and grid middleware: a Medical Data Manager. In *HealthGrid conference (HealthGrid'06)*, pages 14–24, Valencia, Spain, June 2006. IOS Press.

[17] S. Mosser, M. Blay-Fornarino, and M. Riveill. Web Services Orchestration Evolution : A Merge Process For Behavioral Evolution. In *2nd European Conference on Software Architecture (ECSA'08)*, Paphos, Cyprus, Sept. 2008. Springer LNCS.

[18] P.-A. Muller, F. Fleurey, and J.-M. Jézéquel. Weaving executability into object-oriented meta-languages. In *Proc. of MODELS/UML'2005*, LNCS, Jamaica, 2005. Springer.

[19] C. Nemo, T. Glatard, M. Blay-Fornarino, and J. Montagnat. Merging overlapping orchestrations: an application to the bronze standard medical application. In *International Conference on Services Computing (SCC 2007) AR=20*, pages 364–371, Salt Lake City, Utah, USA, July 2007. IEEE Computer Engineering.

[20] M. P. Papazoglou and W. J. V. D. Heuvel. Service oriented design and development methodology. *Int. J. Web Eng. Technol.*, 2(4):412–442, 2006.

[21] C. Pautasso, T. Heinis, and G. Alonso. Jopera: Autonomic service orchestration. *IEEE Data Engineering Bulletin*, 29, September 2006 2006.

[22] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.

[23] N. Russell, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. Workflow data patterns: Identification, representation and tool support. In L. M. L. Delcambre, C. Kop, H. C. Mayr, J. Mylopoulos, and O. Pastor, editors, *ER*, volume 3716 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2005.

[24] J. Tatemura, A. Sawires, O. Po, S. Chen, K. Candan, D. Agrawal, and M. Goveas. Mashup Feeds:: continuous queries over web services. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1128–1130. ACM Press New York, NY, USA, 2007.

[25] W3C. Web service glossary. Technical report, W3C, 2004.

[26] J. Wong and J. I. Hong. Making mashups with marmite: towards end-user programming for the web. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1435–1444, New York, NY, USA, 2007. ACM Press.